REMARKS

Reconsideration and allowance of the subject application are respectfully requested.

As an initial matter, the Examiner has not yet indicated consideration of the documents that were submitted in the Information Disclosure Statement filed on March 31, 2004. Applicant requests consideration of the submitted information and a copy of the initialed PTO-1449 forms.

The Examiner requests formal drawings. Formal drawings are attached with this response. Withdrawal of the objection to the drawings is respectfully requested.

The Examiner requests a copy of the textbook cited on page 9, lines 10-12. Applicant has attempted to obtain a copy of this textbook and discovered that only edition 2 is available at convenient book sellers, and edition 1 is what the Examiner requested. Efforts to obtain a copy of edition 1 are ongoing.

Claims 1, and 7-14 stand rejected under 35 U.S.C. §102(e) as being anticipated by U.S. Patent 6,513,156 to Bak. This rejection is respectfully traversed.

Claims 1 and 7-14 stand rejected under 35 U.S.C. §102(e) as being anticipated by U.S. Patent 6,513,156 to Bak. This rejection is respectfully traversed.

To establish that a claim is anticipated, the Examiner must point out where each and every limitation in the claim is found in a single prior art reference. *Scripps Clinic & Research Found. v. Genentec, Inc.*, 927 F.2d 1565 (Fed. Cir. 1991). Every limitation contained in the claims must be present in the reference, and if even one limitation is

missing from the reference, then it does not anticipate the claim. *Kloster Speedsteel AB v. Crucible, Inc.*, 793 F.2d 1565 (Fed. Cir. 1986). Bak fails to satisfy this rigorous standard.

Bak discloses that particular bytecodes or bytecode sequences are selected to be compiled as a corresponding sequence of native machine instructions (a "snippet"). In column 7, lines 28-33, Bak states that the snippet is executed by executing a new "go-native" virtual machine instruction that replaces or overwrites the initial virtual machine instruction of the selected function portion. The go-native bytecode references (e.g., has a pointer to) the snippet (see column 8, lines 43-44). Snippets are generated during program execution (column 9, lines 37-39) and are held in a separate memory space called the snippet zone (column 9, lines 54-55). The interpreter generates a snippet for the sequence selected for replacement and overwrites the first three bytes of the sequence with the go-native bytecode and a two-byte number specifying the snippet (column 7, lines 54-62).

The Examiner asserts that column 7, lines 12-67 and column 8, lines 1-58 of Bak disclose the features specified in integer (vii) of claim 1 and integer (ii) of claim 14. The Examiner equates the claimed slow-form instruction with the original virtual machine instruction and the claimed fast-form instruction with the new virtual machine instruction, i.e., the "go-native" instruction. The Examiner equates the claimed data store with the bytecode table shown in Figure 13 maintained by the system to store information about the various bytecodes (column 12, lines 53-54). The bytecode table includes "stop"

snippet flags 1059 that indicate whether the bytecode should terminate snippet generation when that bytecode is encountered. Pointers 1061 to the bytecode-specific snippet code may also be stored in the bytecode table.

Contrary to the Examiner's assertion, Bak does not disclose that upon reading an original virtual machine instruction (selected for replacement), the instruction interpreter *checks for* a corresponding "go-native" (fast-form) instruction in the bytecode table (data store) and then executes the "go-native" instruction instead of the original virtual machine instruction. It is this check operation missing from Bak, that provides several advantages described below. Further, although Bak indicates at column 5, lines 2-3 that the data processing system could include a cache, Bak does not disclose that the snippet zone/template table (which Examiner equates with the instruction store) is stored in an instruction store that is *distinct from* main memory, such that it can be accessed by an instruction store port of the process core. Bak also does not disclose that the bytecode table (which Examiner equates with the data store) is stored in an instruction store that is *distinct from* main memory, such that it can be accessed by a data store port of the processor core. Lacking multiple features recited in claims 1 and 14, the anticipation rejection based on Bak should be withdrawn.

Certain dependent claims are rejected for obviousness based on Bak. But Bak fails to suggest the combination of features recited in the independent claims. These claims are directed to the problem of reliably replacing slow-form instructions with fast-form instructions in data processing systems having separate instruction stores and data

stores. The instruction interpreter checks, upon encountering a slow-form instruction, whether a corresponding fast-form instruction exists in the data store, and if present, replaces the slow-form instruction with the fast-form instruction. The additional overhead associated with the check is compensated for by the improved reliability of the instruction replacement. The replacement reliability can be compromised where, for example, the instruction store is read-only and the writing of a modified form of instruction to the data store requires either a flush and reload of some portions of the data store or results in a risk of inconsistency due to different forms of a given instruction being concurrently present in both the instruction store and the data store.

Bak teaches away from feature (vii) of claim 1 and feature (ii) of claim 14 where a check is performed for a corresponding fast-form instruction residing in the data store upon encountering a slow-form instruction. In particular, Bak teaches that during the execution of an interpreted program, the interpreter ***dynamically generates*** a snippet for that sequence and then overwrites that sequence with the go-native bytecode and a two-byte number specifying the snippet. See column 7, lines 54-62. The "go-native" bytecode is an unused bytecode selected for use in Bak's technique. Bak does not disclose that the "go-native" instruction is stored in the bytecode table (counterpart of data store). Furthermore, since the snippet code is generated dynamically by the interpreter during execution of the interpreted program, there would be no need to check for the existence of a go-native instruction in the data store prior to execution of the

snippet. Consequently, a skilled person would have no motivation to modify Bak's system to do so.

The application is in condition for allowance. An early notice to that effect is earnestly solicited.

Respectfully submitted,

NIXON & VANDERHYE P.C.

By: _____
John R. Lastova
Reg. No. 33,149

JRL:at
1100 North Glebe Road, 8th Floor
Arlington, VA 22201-4714
Telephone: (703) 816-4000
Facsimile: (703) 816-4100